实验二 生产者-消费者问题

班级： 07112201 学号： 1120220715 姓名： 刘秉致

一、实验目的

独立设计并编写程序实现生产者-消费者问题，加深对进程通信的理解。

二、实验内容（黑体四号字）

1. 在Windows系统下实现生产者-消费者问题。

2. 具体要求如下：

(1) 创建一个包含6个缓冲区的缓冲池，初始为空，每个缓冲区能存放一个长度为10个字符的字符串。

(2) 创建2个生产者进程，要求每个生产者进程：

随机等待一段时间后，向缓冲区中添加数据；

若缓冲区已满，则等待消费者取走数据后再继续添加数据；

重复添加数据12次。

(3) 创建3个消费者进程，要求每个消费者进程：

随机等待一段时间后，从缓冲区中读取数据；

若缓冲区为空，则等待生产者添加数据后再继续读取数据；

重复读取数据8次。

(4) 输出显示每次添加或读取数据的时间，以及缓冲区的映像。

三、程序设计与实现

1.声明需要使用的头文件

#include <windows.h>

#include <iostream>

#include <string>

#include <vector>

#include <thread>

#include <chrono>

#include <cstdlib>

本实验由于需要创建子线程，因此需要引入<thread>库，同时由于使用了WindowsAPI提供的关于信号量和互斥锁的句柄，因此需要引入<windows.h>库

2.定义全局变量

const int BUFFER\_SIZE = 6;//缓冲区大小

const int STRING\_LENGTH = 10;//缓冲单元中字符串长度

const int PRODUCER\_COUNT = 2;//生产者数量

const int CONSUMER\_COUNT = 3;//消费者数量

const int PRODUCE\_TIMES = 12;//每个生产者生产次数

const int CONSUME\_TIMES = 8;//每个消费者消费次数

3.实现缓冲区和信号量句柄

/// @brief 缓冲区

std::vector<std::string> buffer(BUFFER\_SIZE);

int produceIndex = 0;//生产位置索引,采用循环顺序写入

int consumeIndex = 0;//消费位置索引，采用循环顺序读出

HANDLE emptySlots;  // 信号量，表示空槽位

HANDLE fullSlots;   // 信号量，表示满槽位

HANDLE bufferMutex; // 互斥锁，用于保护缓冲区访问

这里只给出信号量的声明，其初始化将在主函数内实现

4.主函数实现

// @brief 主函数，创建进程和信号量

int main() {

    // 初始化随机数种子

    srand((unsigned)time(NULL));

    emptySlots = CreateSemaphore(NULL, BUFFER\_SIZE, BUFFER\_SIZE, NULL);//初值为BUFFER\_SIZE，最大值为BUFFER\_SIZE

    fullSlots = CreateSemaphore(NULL, 0, BUFFER\_SIZE, NULL);//初值为0，最大值为BUFFER\_SIZE

    bufferMutex = CreateMutex(NULL, FALSE, NULL);//互斥锁,初值为FALSE

    std::vector<std::thread> producers, consumers;//线程容器

    for (int i = 0; i < PRODUCER\_COUNT; ++i) {

        producers.emplace\_back(producer, i + 1);//追加生产者线程

    }

    for (int i = 0; i < CONSUMER\_COUNT; ++i) {

        consumers.emplace\_back(consumer, i + 1);//追加消费者线程

    }

    for (auto& th : producers) {

        th.join();//等待线程结束

    }

    for (auto& th : consumers) {

        th.join();//等待线程结束

    }

    CloseHandle(emptySlots);

    CloseHandle(fullSlots);

    CloseHandle(bufferMutex);

    return 0;

}

在这里我们将信号量的初值和最大值进行了设置：

|  |  |
| --- | --- |
| 信号量 | 初值/最大值 |
| bufferMutex | false/- |
| emptySlots | BUFFER\_SIZE/BUFFER\_SIZE |
| fullSlots | 0/BUFFER\_SIZE |

同时，在主函数中，我们为生产者和消费者分别开辟了队列，在每个线程创建后追加入队列并开始执行。

5.生产者功能函数

// @brief 生产者功能函数

/// @param id 进程id

void producer(int id) {

    for (int i = 0; i < PRODUCE\_TIMES; ++i) {

        randomSleep();

        std::string data(STRING\_LENGTH, 'A' + rand() % 26);//生成一个随机大写字母的字符串

        WaitForSingleObject(emptySlots, INFINITE); // 等待空槽

        WaitForSingleObject(bufferMutex, INFINITE); // 获取互斥锁

        buffer[produceIndex] = data;

        std::cout << "["<< getCurrentTime() << "][Producer" << id << "] Produced: "

        << data<< " at index " << produceIndex << std::endl;

        printBuffer();

        produceIndex = (produceIndex + 1) % BUFFER\_SIZE;

        ReleaseMutex(bufferMutex); // 释放互斥锁

        ReleaseSemaphore(fullSlots, 1, NULL); // 增加满槽计数

    }

}

生产者将产生一个随机大写字母填充的符合长度的字符串

6.消费者功能函数

/// @brief 消费者功能函数

/// @param id 进程id

void consumer(int id) {

    for (int i = 0; i < CONSUME\_TIMES; ++i) {

        randomSleep();

        WaitForSingleObject(fullSlots, INFINITE); // 等待满槽

        WaitForSingleObject(bufferMutex, INFINITE); // 获取互斥锁

        std::string data = buffer[consumeIndex];

        buffer[consumeIndex] = "";

        std::cout <<"["<< getCurrentTime() << "][Consumer " << id << "] Consumed: " << data

                  << " at index " << consumeIndex << std::endl;

        printBuffer();

        consumeIndex = (consumeIndex + 1) % BUFFER\_SIZE;

        ReleaseMutex(bufferMutex); // 释放互斥锁

        ReleaseSemaphore(emptySlots, 1, NULL); // 增加空槽计数

    }

}

消费者读出缓冲区内字符串。

6.其他功能函数

/// @brief 随机休眠函数

void randomSleep() {

    std::this\_thread::sleep\_for(std::chrono::milliseconds(rand() % 1000 + 500));

}

/// @brief 获取当前时间

std::string getCurrentTime() {

    auto now = std::chrono::system\_clock::now();

    auto time = std::chrono::system\_clock::to\_time\_t(now);

    auto ms = std::chrono::duration\_cast<std::chrono::milliseconds>(

        now.time\_since\_epoch()) % 1000;

    char buffer[26];

    ctime\_s(buffer, sizeof(buffer), &time);

    std::string timeStr(buffer);

    timeStr = timeStr.substr(11, 8) + "." +

              std::to\_string(ms.count());

    return timeStr;

}

///@brief 显示缓冲区状态

void printBuffer() {

    std::cout << "Buffer status: [";

    for (int i = 0; i < BUFFER\_SIZE; ++i) {

        if (buffer[i].empty()) {

            std::cout << "Empty";

        } else {

            std::cout << buffer[i];

        }

        if (i < BUFFER\_SIZE - 1) std::cout << " | ";

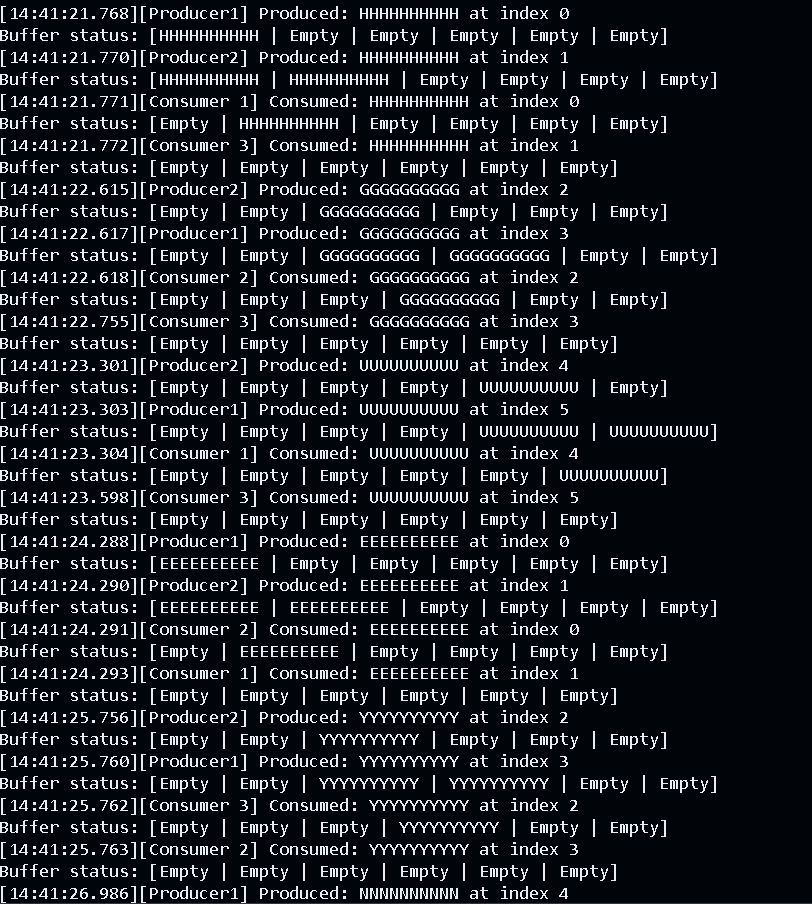
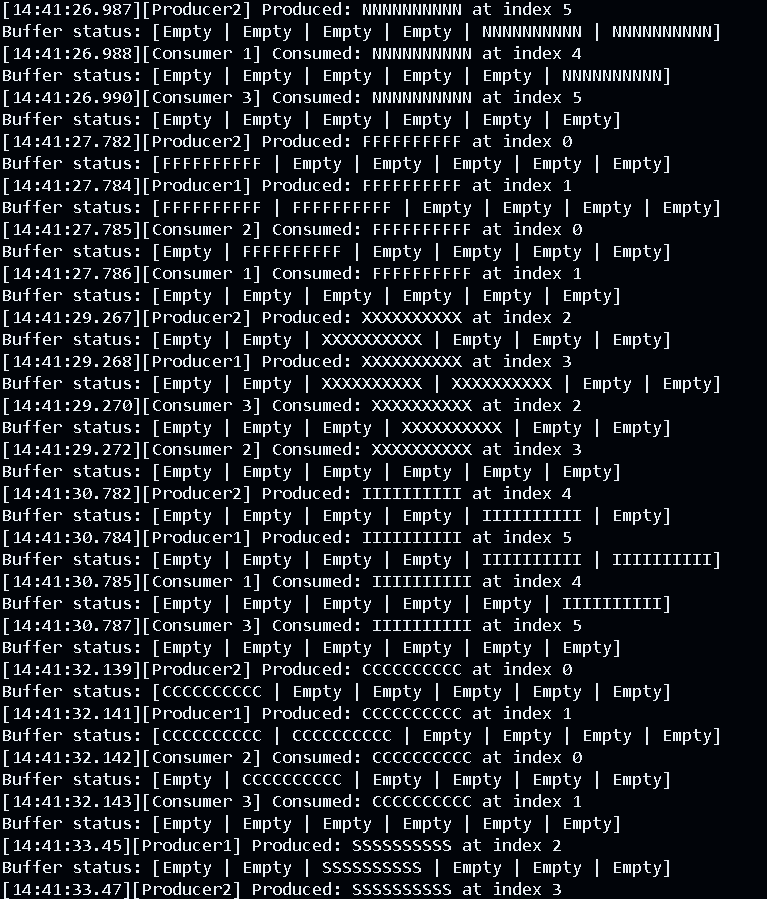
    }

    std::cout << "]" << std::endl;

}

四、实验结果及分析

在Window系统终端Powershell下执行，观察程序正常执行，产生了符合要求的生产者子线程和消费者子线程。同时可以观察到生产者和消费者几乎交替的对缓冲区进行读写操作，体现了两者竞争读写锁的过程。



五、实验收获与体会

经过本次实验，我更加深入的了解了系统中的进程之间该如何基于信号量进行通信，实现互斥或协作。

除此之外，这是我第一次在C语言程序中调用WindowsAPI实现多线程并行，这对我来说是一次新的体验，学习了如何显式的在程序中实现多线程开发。

附录：程序清单及说明

1. PV.cpp 该实验的源代码